

Remarks

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks. Claims 1-10 and 20 are pending the application. Claims 1 and 20 are independent. New claims 21-27 have been added. Claims 21-23 depend on claim 20. claims 24-26 depend on claim 1. Claim 27 is an independent claim.

Cited Art

Paged virtual memory schemes described in the background section of the present application and U.S. Pat. No. 5,611,064 to *Maund* et al. ("*Maund*").

Patentability of Claim 20 over prior paged virtual memory schemes under 35 U.S.C. § 102(b)

The action rejects claim 20 under 35 U.S.C. § 102(b) as unpatentable over paged virtual memory schemes, such as described in the background section of the present application at page 4, line 10 – page 9, line 2. Applicants respectfully submit claim 20 in its present form is patentable over the cited art.

As amended, claim 20 is directed to a computer readable medium comprising a data structure to be processed for virtual memory management. More particularly, claim 20 recites as follows:

A computer-readable medium having stored thereon a data structure comprising:

a series of data fields forming a group for indicating blocks of code or data associated with an application to be treated as a single unit for purposes of virtual memory management, the data fields including a list of memory addresses of the blocks and sizes of each block in the list;

wherein the data structure is evaluated in a data processing operation to load each of the blocks into physical memory whenever a not-present interrupt is generated for any memory address referring to a location included in one of the blocks.

The paged virtual memory scheme described in the present application's background section for mapping virtual memory addresses to physical addresses does not teach or suggest "a series of data fields forming a group for indicating blocks of code or data associated with an application to be treated as a single unit for purposes of virtual memory management, the data

fields including a list of memory addresses of the blocks and sizes of each block in the list.” In the recited arrangement, multiple blocks of code comprising multiple pages can be grouped together using “a series of data fields” which would allow all blocks in the group to be processed “as a single unit for the purpose of virtual memory management.” For instance, the specification at page 14, line 21 - page 15, line 6 describes such ‘a series of data fields’ as follows:

To create a group, the application invokes a function in the API implementation 164 (called CreateGroup) to create a data structure for maintaining a list of the pieces of code and data in the group. The application can specify sections of code or data to be placed in the group as it is being created. The application specifies the code and data to be placed in the group by providing the address and size of the sections of virtual memory used to store the code and data. In this implementation, for example, the application provides an array of pointers to blocks of memory to be placed in the group and an array of parameters that provide the sizes of the blocks.

In response to the request to create the group, the API implementation creates a data structure 190 listing all of the sections of memory. In the example in Fig. 4, the data structure includes a list of four blocks of memory. The address for each block points to a location in virtual memory where the block resides (in this case, the marked sections 172, 174, 176 and 178). The data structure 190 also keeps track of the size of each block of virtual memory in the group.

From this data structure 190, the API implementation derives a list of the units of memory corresponding the code or data in the group. (Emphasis added).

The Action relies on the paged virtual memory scheme described in the background section, which is entirely different from the recited “series of data fields.” First of all, the 32-bit virtual address relied on by the Action is not “a series of data fields forming a group for indicating blocks of code or data associated with an application to be treated as a single unit for purposes of virtual memory management.” Instead, the 32-bit address simply indicates an address for a specific page within the virtual memory space which is used to map the virtual address location to a location in the physical memory space. Thus, the 32-bit address does not teach or suggest “data fields including a list of memory addresses of the blocks and sizes of each block in the list.” The specification, at page 13, lines 23-24 describes “memory addresses of the blocks and sizes of each block in the list” as follows:

A block can be defined by a starting address of a portion of virtual memory and its size.

The data fields of the 32-bit memory address do not include such “memory addresses of the blocks.” Furthermore, “a block, in this context, is a set of pages.” (See e.g., specification

page 17, line 11). Nothing in the present paged virtual memory scheme teaches or suggests “a series of data fields forming a group for indicating blocks of code or data associated with an application to be treated as a single unit for purposes of virtual memory management, the data fields including a list of memory addresses of the blocks and sizes of each block in the list” wherein “a block...is a set of pages.” This is so because the 32-bit addresses were used for simply mapping memory references in virtual memory space to addresses in physical memory space. Thus, they do not teach or suggest “a series of data fields forming a group for indicating blocks of code or data associated with an application to be treated as a single unit for purposes of virtual memory management.”

In fact, the methods and data structures of the paged virtual memory scheme can be used in conjunction with the recited “series of data fields.” For instance, once the recited “series of data fields forming a group for indicating blocks of code or data associated with an application to be treated as a single unit for purposes of virtual memory management” has been created if any of the pages within the grouped “blocks of code or data” are accessed then using the methods of the paged virtual memory scheme, including the 32-bit addresses, the processor can ensure that all the pages of all the “blocks of code and data” within the group are loaded onto to physical memory. The specification clearly describes the use of the recited “series of data fields,” at page 12, line 29 – page 13, line 4, as follows:

More specifically, the API includes functions that enable applications to group their code and data together so that the code and data in the group is loaded into physical memory together. To implement this feature, a virtual memory manager keeps track of code and data in a group. Whenever an application tries to access an instruction or data structure in the group, the virtual memory system will load the entire group of code and data into physical memory. (Emphasis added).

As evidenced by the passage above, the grouping of “blocks of code and data” as indicated by the “series of data fields forming a group for indicating blocks of code or data associated with an application to be treated as a single unit for purposes of virtual memory management” is separate and different from the process of loading of such grouped “code or data” into physical memory which is where the methods and systems of the paged virtual memory scheme are applicable. Thus, the paged virtual memory scheme and the associated 32-bit addressing scheme fail to teach or suggest “a series of data fields forming a group for indicating blocks of code or data associated with an application to be treated as a single unit for

purposes of virtual memory management, the data fields including a list of memory addresses of the blocks and sizes of each block in the list.” Thus, at least for this reason claim 20 in its present form is patentable over the paged virtual memory scheme. Claim 20 should therefore be allowed. Such action is respectfully requested.

Patentability of Claims 1-10 over *Maund* under 35 U.S.C. § 102(b)

The action rejects claims 1-10 under 35 U.S.C. § 102(b) as unpatentable over *Maund*. Applicants respectfully submit claims 1-10 in their present form are patentable over the cited art.

Claim 1

Claim 1 in its present form is directed to a method of allowing an application program to control the allocation of physical memory in a multitasking operating system. More particularly, claim 1 recites as follows:

In a multitasking operating system that uses virtual memory to share physical memory among concurrently executing application programs, a method for controlling allocation of physical memory comprising:

- in response to a call from an application program, other than an operating system, to group specified code or data in a group, creating a structure to group the code or data specified by the application;
- monitoring for a not-present interrupt generated in response to request to access any part of the code or data in the group; and
- when the not-present interrupt occurs for a unit of memory in the group, loading all of the code or data in the group that is not already in physical memory into physical memory from secondary storage at one time, including loading the unit of memory for which the not-present interrupt has occurred and all other units of memory used to store the code or data in the group.

The pre-determined and inflexible grouping of patches, pages and superpages of Maund do not teach or suggest “in response to a call from an application program, other than an operating system, to group specified code or data in a group, creating a structure to group the code or data specified by the application.” The recited method allows an “application program, other than an operating system, to group specified code or data in a group” for the purposes of conducting virtual memory management including “when the not-present interrupt occurs for a

unit of memory in the group, loading all of the code or data in the group that is not already in physical memory into physical memory from secondary storage at one time.”

The Action relies on *Maund*, which teaches methods and systems for “providing address translation means which does not require tables of such massive size.” (*See, Maund* at Col. 2, lines 4-6). Whereas *Maund* is concerned with reducing the size of page tables required for translating virtual memory addresses to physical memory addresses, the subject claim 1 is directed to giving greater control to application programs over “how a virtual memory system allocates physical memory.” (*See, Specification* page 12, lines 28-29). Thus, *Maund* has no need for and does not teach or suggest “in response to a call from an application program, other than an operating system, to group specified code or data in a group, creating a structure to group the code or data specified by the application.”

Maund does teach or suggest “the pages are arranged in the virtual memory space in groups.” (*See, Maund* Abstract). However, *Maund* does not teach or suggest such grouping of pages is “in response to a call from an application program, other than an operating system, to group specified code or data in a group.” In fact, the grouping of pages taught by *Maund* is pre-determined and inflexible and thus, not “in response to a call from an application program, other than an operating system, to group specified code or data in a group.” For instance, *Maund* teaches or suggests that the units of virtual memory are grouped not “in response to a call from an application program, other than an operating system, to group specified code or data in a group” but instead, in a pre-determined manner, “virtual memory address space is considered to be formatted as aligned groups of the pages, with *the pages in each group being aligned and contiguous.*” (*See, Maund* at Col. 2, Lines 8-10) (Emphasis added). This inflexibility of *Maund*

is indicative of the fact that Maund fails to teach or suggest “in response to a call from an application program, other than an operating system, to group specified code or data in a group.”

This inflexibility of *Maund* is further illustrated clearly in Figs. (6A-B and 7A-B) of *Maund*, which shows grouping of pages as being pre-determined and restricted to consecutive pages of a virtual memory space. On the other hand, Fig. 4 of the specification shows that grouping of pages into blocks (*e.g.*, 172, 174, 176, and 180) is not restricted to consecutive pages. This is not to say that only non-consecutive pages can be grouped in accordance with the claimed method. In fact, “in response to a call from an application program, other than an operating system, to group specified code or data in a group” any and all pages of a virtual memory space can be grouped into blocks and blocks can then be grouped to form groups. Thus, groups can be formed of consecutive pages or non-consecutive pages. The relevant distinction of claim 1 from *Maund* is that *Maund* does not teach or suggest grouping of pages “in response to a call from an application program, other than an operating system, to group specified code or data in a group.” This is an important difference because using the recited method allows application programmers greater control over “how a virtual memory system allocates physical memory.” (*See*, Specification page 12, lines 28-29). On the other hand this is not taught or suggested by *Maund*.

Claim 1 therefore recites at least one element not taught or suggested by the cited art. Thus, at least for this reason claim 1 in its present form is patentable over *Maund*. Claim 1 should therefore be allowed. Such action is respectfully requested.

Claims 2-4 and 7-10

Claims 2-4 and 7-10 depend on claim 1. Thus, at least for the reasons set forth above with regard to claim 1, claims 2-4 and 7-10 should be patentable.

Claims 5-6

Claims 5-6 depend on claim 1. Thus, at least for the reasons set forth above with regard to claim 1, claims 5-6 should be patentable. However, the Applicants would like to point out that the rejection of claims 5-6 under 35 U.S.C. § 102(b) is improper at least to the extent this rejection relies on multiple references. (*See*, Action at page 6-7 relying on both the paged virtual memory described in the background section at page 14, line 21 - page 15, line 6 and *Maund* to reject claims 5 and 6.) Typically, “a claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference.” *Verdegaal Bros. v. Union Oil Co. of California*, 814 F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987) (Emphasis added). None of the noted exceptions to this general rule are alleged in this Action. (*See e.g.*, MPEP § 2131.01) (discussing multiple reference 35 U.S.C. § 102 rejections). Thus, the rejections of claims 5-6 under 35 U.S.C. § 102(b) should be withdrawn.

As noted above, claims 5-6 depend on claim 1. Also, neither *Maund* nor the paged virtual memory scheme, either by themselves or in combination thereof, teach or suggest “in response to a call from an application program, other than an operating system, to group specified code or data in a group, creating a structure to group the code or data specified by the application.” Thus, based on the arguments presented above with regard to claim 1, claims 5-6

are patentable over *Maund* and the paged virtual memory scheme either by themselves or a combination thereof.

Patentability of new claims 21-27

New claims 21-27 have been added. Claims 21-23 depend on the currently amended claim 20. Thus, at least for the reasons set forth above with regard to claim 20, claim 21-23 should also be patentable. Claims 24-26 depend on claim 1. Thus, at least for the reasons set forth above with regard to claim 1, claims 24-26 should also be patentable. Claim 27 is a means plus function claim relating to the method of claim 1. Thus, at least for the reasons set forth above with regard to claim 1, claim 27 should also be patentable.

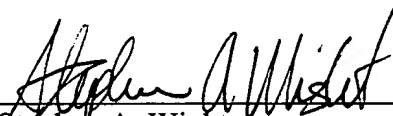
Conclusion

The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

By



Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446